Patent Application for
Kevin W Jameson

Collection Extensible Action GUI

Cross Reference To Related Applications

The present invention uses inventions from the following patent applications, which are incorporated herein by reference:

Collection Information Manager, USPTO Patent Application 09/885078, filed June 21, 2001, Kevin W Jameson.

Collection Knowledge System, USPTO Patent Application 09/885079, filed June 21, 2001 Kevin W Jameson.

Collection Role Changing GUI, USPTO Patent Application 10/003,423 filed December 6, 2001, Kevin W Jameson.

Field of the Invention

This invention relates to graphical user interfaces for processing collections of computer files in arbitrary ways, thereby improving the productivity of software developers, web media developers, and other humans that work with collections of computer files.

buttons on a GUI interface. Work operations are typically implemented as independent command line scripts or programs, or as subroutine calls within GUI interface programs.

Dominant CLI Design Strategies

Simply put, there are no dominant CLI design strategies. All CLI interfaces are essentially the same---users type command lines into a CLI window, and the operating system executes the command line. Although command line I/O models (pipes, tees, redirections) from the 1970s were novel, it is substantially fair to say that CLI interfaces have not really changed much in the past several decades.

Dominant GUI Design Strategies

In contrast to CLI interfaces, GUI interfaces have evolved significantly during the past 30 years.

For several decades now, the dominant GUI interface design strategy has been to write a unique GUI interface for each distinct user application domain. For example, unique GUI interfaces have been implemented for spreadsheet programs, word processing programs, email programs, and database programs.

The "one GUI per application domain" design strategy makes sense because each unique GUI interface provides users with a custom set of GUI work operations that are related to the data and operations used within a particular application domain. As a counter example, to illustrate the point again, it makes little sense to provide spreadsheet buttons on a word processing interface, or to provide word processing buttons on a database interface.

The "one GUI per application domain" design strategy also makes sense from a marketing point of view, because a unique GUI interface can be more easily differentiated from other products in the marketplace.

A second part of the dominant GUI design strategy provides a fixed set of work operations for each unique GUI interface. To build an interface, GUI designers study

executed as part of performing a requested action. Multiple action commands may be contained within a single action.

An Action Data Storage Means is a data storage means that stores GUI action data outside the GUI itself. An action data storage means can generally store all data used by a Collection Extensible Action GUI. Stored GUI action data can be created, accessed, and shared by multiple human users without using compilers or other special GUI programming tools. For example, three preferred action storage means are ASCII files, relational databases, and Collection Knowledge Systems (see the section on related patent applications for more information on Collection Knowledge Systems).

A Context Sensitive Action Data Storage Means is an action data storage means that can return different answers to data lookup queries, depending on the value of a context token provided as part of the data query. Collection Knowledge Systems are context sensitive.

Extensible GUI Action Data

The intent of this material is to provide readers with an overview of how GUI actions can be modeled by simple, user-defined action data files.

This section is an introduction to the structure and organization of Action Data files. Action Data files model a GUI from the action level (mid-level) through to executable commands (low-level).

In contrast, Role Data files model GUI functionality from the layout level (high-level) to the action level (mid-level). For an introduction to the structure and organization of role data files, see the related patent applications listed in the Related Patent Applications section at the beginning of this document.

Although the examples shown below use simple ASCII files for presentation clarity, readers of ordinary skill in the art will immediately appreciate that the ASCII files shown here could easily be implemented using various, more advanced data storage means, including relational databases.

callback subroutine that executes an internal command.

Actions: Ext rnal Commands

If required, Module Execute Command External 173 is called to oversee the execution of external operating system commands. Various internal subroutines can be called to oversee variations in how external commands are executed.

As one example, FIG 13 Line 14 "callback-xcmd" executes a sequence of external operating system command lines in the current working directory.

As a second example, FIG 13 Line 30 "callback-platform" executes a sequence of external operating system commands from within the platform directory of a collection. For more information on collections, see the section on related patent applications listed at the beginning of this document.

As a third example, FIG 13 Line 22 "callback-fvar" first substitutes focus variable values into a command line template Line 24 and then executes the instantiated command line in the current working directory.

The numbers of, and actions of, internal and external callback subroutines are determined by the design goals of the implementation.

Actions: Focus Variable Operations

If required, Module Execute Focus Variable Operations 174 is called to oversee various string operations (e.g. set, join, split) on pathnames stored in focus variables. Focus variable operations are useful when actions work with pathnames containing both a directory portion and a filename portion.

FIG 14 shows two example actions that use focus variable operations to set and split a focus variable pathname value, respectively.